One for All: Update Parameterized Knowledge Across Multiple Models with Once Edit

Weitao Ma[1*] **Xiyuan Du**[1*] **Xiaocheng Feng**[1,2†] **Lei Huang**[1] **Yichong Huang**[1]
Huiyi Zhang[1] **Xiaoliang Yang**[1] **Baohang Li**[1] **Xiachong Feng**[3] **Ting Liu**[1] **Bing Qin**[1,2]
[1] Harbin Institute of Technology
[2] Peng Cheng Laboratory
[3] The University of Hong Kong
{wtma,xcfeng}@ir.hit.edu.cn

Abstract

Large language models (LLMs) encode vast world knowledge but struggle to stay up-to-date, often leading to errors and hallucinations. Knowledge editing offers an efficient alternative to retraining, enabling targeted modifications by updating specific model parameters. However, existing methods primarily focus on individual models, posing challenges in efficiently updating multiple models and adapting to new models. To address this, we propose ONCEEDIT, a novel ensemble-based approach that employs a plug-in model as the editing module, enabling stable knowledge updates across multiple models. Building on the model ensemble, ONCEEDIT introduces two key mechanisms to enhance its effectiveness. First, we introduce a *dynamic weight mechanism* through a `[WEIGHT]` token for distinguishing between edit-related and non-edit-related instances, ensuring the appropriate utilization of knowledge from integrated models. Second, we incorporate an *ensemble enhancement mechanism* to mitigate the excessive reliance on the central model inherent in the model ensemble technique, making it more suitable for knowledge editing. Extensive experiments on diverse LLMs demonstrate that ONCEEDIT consistently outperforms existing methods while achieving superior editing efficiency. Further analysis confirms its adaptability and stability in multi-model editing scenarios.

1 Introduction

Large language models (Achiam et al., 2023; Jiang et al., 2023a; Meta, 2024) have demonstrated remarkable performance in various downstream tasks by scaling in both parameters and training data, thereby capturing extensive world knowledge during pretraining (Wang et al., 2024a; Feng et al., 2023). However, as real-world information undergoes dynamic changes, the internal parameterized knowledge of LLMs gradually becomes outdated, resulting in errors and hallucinations (Huang et al., 2023; Zhang et al., 2023; Zhong et al., 2024), hindering the practical application of LLMs. Currently, efforts to mitigate hallucinations mainly focus on two aspects: faithfulness (Huang et al., 2024a, 2025) and factuality (Li et al., 2024a). Among these, knowledge editing has emerged as a promising approach for improving the factual accuracy of LLMs by directly modifying their internal knowledge. Rather than resorting to costly retraining, knowledge editing provides an efficient and practical ways to update a model's knowledge (Yao et al., 2023; Zhang et al., 2024a). These techniques enable the integration of growing knowledge into the models by allowing precise updates through the targeted parameters modification (Li et al., 2024b).

In recent years, various knowledge editing methods for LLMs have been proposed, leveraging techniques such as meta-learning (Mitchell et al., 2021; Tan et al., 2023), locate-then-edit strategies (Meng et al., 2022a,b), and memory-based approaches (Mitchell et al., 2022; Hartvigsen et al., 2024) to

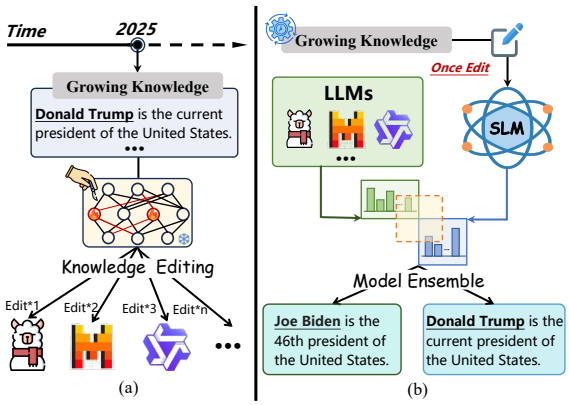

Figure 1: The comparison of traditional knowledge editing and ONCEEDIT for multi-model updates. (a) Traditional methods require separate edits for each model, while (b) ONCEEDIT updates all models with a single edit via model ensemble.

* means Equal Contribution
† means Corresponding Author

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 16021–16034
July 27 - August 1, 2025

update model knowledge while preserving unrelated information. However, existing methods primarily focus on modifying a single model, which makes them unsuitable for complex scenarios requiring the simultaneous update of multiple models. Additionally, these methods exhibit significant sensitivity to hyperparameter settings, leading to considerable inconsistency in editing effectiveness across models, which limits their scalability and adaptability to new models.

To address these challenges, we introduce ONCEEDIT, which modifies a unified lightweight plug-in model and employs a heterogeneous model ensemble for knowledge transfer across multiple models, thereby enabling seamless and stable knowledge editing, as shown in Figure 1. However, model ensemble methods are not directly applicable to editing scenarios, and ONCEEDIT introduces two improvement mechanisms to align more closely with knowledge editing tasks. Firstly, traditional ensemble methods fuse the knowledge of the plug-in model and the LLM using fixed ensemble weights, making them unsuitable for knowledge editing, where new knowledge should be updated without affecting unrelated information. To this end, ONCEEDIT introduces a *dynamic weight mechanism* using a special `[WEIGHT]` token, which predicts weight allocation for each instance, ensuring the effective utilization of knowledge from integrated models. Secondly, model ensembles often suffer from the inherent bias of the central large model, where its knowledge dominates the ensemble results compared to the plug-in model. To counter this, we propose an *ensemble enhancement mechanism* that incorporates two strategies: *search-space zero initialization* and *target augmentation*. By starting the decoding search with a zero vector instead of the central model's distribution, and emphasizing high-probability tokens from the fused distribution, these strategies ensure that the decoding is driven by the fused knowledge, improving both the precision and generalization of the edited knowledge.

We conduct extensive experiments on Llama2-7B, Mistral-7B-v0.1, and GPT-J-6B using the ZsRE and Counterfact datasets to compare the performance of ONCEEDIT against seven popular knowledge editing methods. Experimental results demonstrate that ONCEEDIT consistently outperforms other methods in both teacher-forced and validation generation evaluation settings, which better align with realistic scenarios, while also requiring the fewest editing interventions. Additionally, we quantitatively analyze the editing time of each method, showing that ONCEEDIT incurs the lowest editing overhead in multi-model knowledge editing scenarios. Furthermore, we extend our evaluation to more and larger models, such as Qwen2.5-7B, Llama3-70B, etc., to further validate the adaptability and stability of ONCEEDIT.

2 Preliminaries

In this section, we introduce knowledge editing as the core task of our study and model ensemble as the underlying technique supporting our methods.

2.1 Knowledge Editing

Knowledge editing is an effective technique for updating LLMs with new knowledge. Given a target model that is parametrized by θ and a new edited set S_E, the goal of knowledge editing is to update the model so that it correctly responds to the edits while maintaining its unrelated knowledge. The knowledge editing function, denoted as $\text{KE}(\theta, S_E)$, represents the process of modifying the model θ based on the edited knowledge set S_E. The editing process can be expressed as follows:

$$\theta' \leftarrow \text{KE}(\theta, S_E), \tag{1}$$

let $f_\theta(\cdot)$ represent the original mapping function of the model θ. The expected output of the edited model θ' is defined as follows:

$$f_{\theta'}(x) = \begin{cases} y_e & \text{if } x \in I_{edit}, \\ f_\theta(x) & \text{otherwise.} \end{cases} \tag{2}$$

Here, I_{edit} represents the set of instances within the editing scope of the edits in S_E. In addition to S_E, I_{edit} may also include knowledge-related input, such as re-phrased versions of the edit input.

Following previous research (Wang et al., 2024b; Zhang et al., 2024a), an ideal knowledge editing method should ensure that the edited model meets three key properties: **Reliability**, **Generality**, and **Locality**. These properties collectively ensure that the edited model maintains correctness on targeted updates, generalizes appropriately, and preserves unaffected knowledge. Details about these three properties can be found in Appendix B.

2.2 Model Ensemble

Existing knowledge editing methods primarily focus on single models, making it difficult to efficiently adapt across models. This challenge motivates us to propose a model ensemble framework,

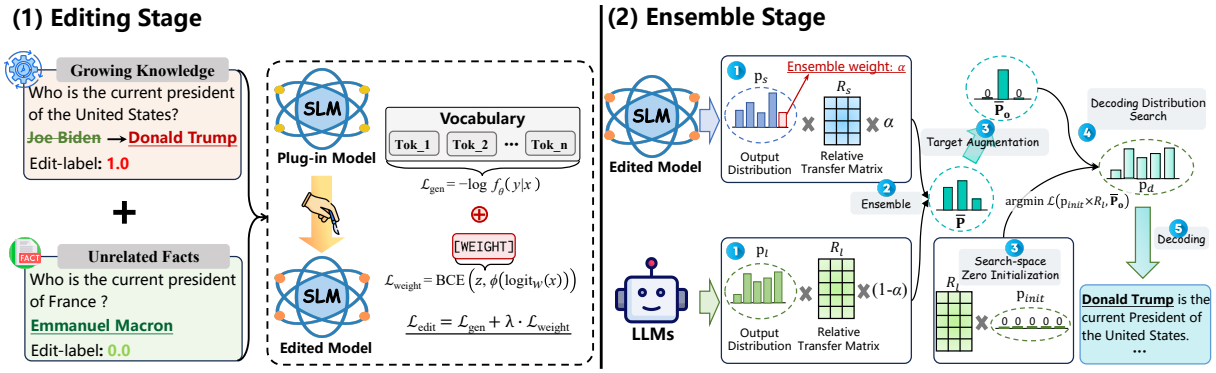

Figure 2: Overview of OnceEdit, which consists of two stages. In the editing stage, OnceEdit applies knowledge edits to a lightweight model while introducing [WEIGHT] to learn the ensemble weights (§3.1). In the ensemble stage, the edited model is integrated with LLMs to achieve multi-model knowledge updating (§3.2).

where a small plug-in model serves as an edited module to reliably modify various LLMs.

The model ensemble techniques integrate the output distributions from multiple models to achieve an optimal result. Specifically, the aggregated output probability, denoted as $\overline{\mathbf{P}}$, is calculated as the sum of the weighted probabilities of each model, where α_i represents the weight assigned to the i-th model and $\mathbf{p}_i$ is the output probability distribution from the i-th model. The ensemble process can be expressed as follows:

$$\overline{\mathbf{P}} = \sum_{i=1}^{N} \alpha_i \times \mathbf{p}_i. \tag{3}$$

However, when the candidate models to be integrated are heterogeneous, additional steps are needed to align their vocabularies before effective integration. For instance, in the classic heterogeneous model ensemble method, DEEPEN (Huang et al., 2024b), the procedure involves selecting a set of common tokens shared across models to serve as anchor words. The distance between other words and each anchor word is then computed, resulting in a relative transfer matrix. This matrix is used to map each model's output probability into the relative representation space, facilitating the integration of their probability distributions. The process can be expressed as follows:

$$\overline{\mathbf{P}} = \sum_{i=1}^{N} \alpha_i \times (\mathbf{p}_i \times R_i), \tag{4}$$

where $R_i \in \mathbb{R}^{|V_i| \times |A|}$ represents the relative transfer matrix of the i-th model. Here, $|V_i|$ denotes the corresponding vocabulary size of the i-th model and $|A|$ represents the number of anchor words.

3 Methodology

In this section, we introduce OnceEdit based on DEEPEN (Huang et al., 2024b), an effective method for integrated heterogeneous models. The overall process of OnceEdit comprises two stages: *Editing Stage* and *Ensemble Stage*, as shown in Figure 2. In the editing stage, OnceEdit selects a lightweight plug-in model as the editing module, which is updated using knowledge editing techniques to incorporate new information (§3.1). To enhance DEEPEN's static weight allocation, OnceEdit introduces the **Dynamic Weight** echanism, which enables instance-level weight adjustment. In the ensemble stage, the edited model is integrated with multiple LLMs for knowledge updating (§3.2). To better align DEEPEN with knowledge editing, OnceEdit incorporates the **Ensemble Enhancement** mechanism, leveraging *Search-space Zero Initialization* and *Target Augmentation* to stabilize the new knowledge transfer.

3.1 Editing Stage

To efficiently facilitate multi-model knowledge updates, editing the plug-in model plays a crucial and foundational role. In this context, OnceEdit employs a simple but effective full fine-tuning strategy to update the knowledge within the plug-in model. This approach is particularly well-suited since the plug-in model is relatively small, keeping the associated computational cost manageable. The following outlines the training objectives:

$$\mathcal{L}_{\text{gen}}(\theta) = -\mathbb{E}_{(x,y)\in S_E} \left[\log f_\theta\left(y \,|x\right)\right]. \tag{5}$$

However, full fine-tuning often results in significant degradation of the model's original knowledge. To address this issue, OnceEdit introduces

a **Dynamic Weighting** mechanism that adaptively adjusts the contribution of each model during the ensemble stage based on the given input. Specifically, we introduce a special token, `[WEIGHT]`, into the vocabulary of the plug-in model. This token helps distinguish between knowledge that requires modification and knowledge that should remain unaffected. Consequently, for edit-related inputs, the plug-in model is assigned a higher weight, whereas for non-edit-related inputs, the LLMs dominate.

To effectively train `[WEIGHT]`, it is essential not only to fine-tune the model on the target knowledge modifications but also to introduce a set of unrelated knowledge as a reference group to guide the model in distinguishing between edit-related and non-edit-related knowledge. The training objective of the token is formulated as follows:

$$\mathcal{L}_{\text{weight}}(\theta) = \mathbb{E}_{(x,z)}\text{BCE}\left(z, \phi(\text{logit}_w(x))\right), \quad (6)$$

where z denotes the edit-label of the input, where instances related to edits are assigned a value of 1, and instances not related to edits are assigned a value of 0. Additionally, $\text{logit}_w(x)$ represents the logits at the position of the `[WEIGHT]` after encoding the input x. In this context, $\phi(\cdot)$ and $\text{BCE}(\cdot)$ denote the sigmoid function and the binary cross-entropy loss function, respectively.

Finally, we adopt a multi-task learning approach to jointly train the plug-in model. The overall training objective of the editing stage is formulated as:

$$\mathcal{L}_{\text{edit}}(\theta) = \mathcal{L}_{\text{gen}}(\theta) + \lambda \cdot \mathcal{L}_{\text{weight}}(\theta), \quad (7)$$

where λ is a hyperparameter that balances the learning contributions of the two tasks.

3.2 Ensemble Stage

During the ensemble stage, as previously described in §2.2, we select the common words shared between the plug-in model and the LLMs as anchor words and calculate the corresponding relative transfer matrix. At each decoding step, the ensemble models use the corresponding relative transfer matrices to map the output distribution into the relative space, where it is then fused with weighted contributions. The aggregated distribution is then obtained by combining the outputs, with the weight provided by `[WEIGHT]`:

$$\overline{\mathbf{P}} = \alpha \times (\mathbf{p}_s \times R_s) + (1-\alpha) \times (\mathbf{p}_l \times R_l), \quad (8)$$

where $\mathbf{p}_s$ and $\mathbf{p}_l$ represent the output distributions of the plug-in model and the LLMs. R_s and R_l are the relative transfer matrices for these models. $\alpha = \phi(\text{logit}_w(x))$ is the ensemble weight derived from the plug-in model.

Once the aggregated distribution is obtained, we use the LLM as the decoding model. Following the DEEPEN framework, we employ gradient descent to search for an optimal output distribution within the vocabulary space of the LLM, ensuring that the aggregated distribution is accurately represented. The process is formalized as follows:

$$\mathbf{p}_d = \arg\min \mathcal{L}(\mathbf{p}_{init} \times R_l,\ \overline{\mathbf{P}}), \quad (9)$$

where $\mathbf{p}_{init}$ and $\mathbf{p}_d$ denote the initial search distribution and the final decoding distribution, both in the absolute representation space of the LLMs.

DEEPEN originally initializes the search using the LLM's output distribution $\mathbf{p}_{init} = \mathbf{p}_l$, treating the aggregated distribution as a perturbation to the LLM's original output. This approach is effective for traditional model ensemble, where the integrated models produce similar outputs, allowing for minor corrections for LLM's behaviors. However, this method can lead to the central model becoming biased, resulting in the ensemble's output being overly reliant on the LLM's knowledge. In the context of knowledge editing, where the plug-in model and the LLM often exhibit significant distributional differences, using the LLM's original distribution for initialization may fail to effectively capture the newly injected knowledge.

Based on the above analysis, we propose an **Ensemble Enhancement** mechanism including two strategies: *Search-space Zero Initialization* and *Target Augmentation* to better align the decoding distribution with the aggregated distribution. These strategies work together to strengthen the search process, as described below:

$$\mathbf{p}_{init} = \text{zeros\_like}(\mathbf{p}_l), \quad (10)$$

$$\overline{\mathbf{P}_\mathbf{o}} = \begin{cases} 1, & i = \arg\max_j \overline{\mathbf{P}}_j, \\ 0, & \text{otherwise}, \end{cases} \quad (11)$$

where $\text{zeros\_like}(\cdot)$ is a function that creates a vector with the same shape as the input, but with all elements set to 0.

These two strategies work as follows: the first, Search-space Zero Initialization, initializes the search space with a zero vector. The second, Target Augmentation, converts the aggregated distribution into a one-hot vector. Together, these strategies help the final decoding distribution better capture

Method	Llama2-7B				Mistral-7B-v0.1				GPT-J-6B				Score↑	Freq.↓
	Rel.↑	Gen.↑	Loc.↑	Avg.↑	Rel.↑	Gen.↑	Loc.↑	Avg.↑	Rel.↑	Gen.↑	Loc.↑	Avg.↑		
ZsRE														
FT-L	0.34	0.21	0.13	0.23	0.55	0.41	0.54	0.50	0.11	0.10	0.48	0.23	0.32	3
MEND	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	3
ROME	0.07	0.06	0.01	0.05	0.01	0.01	0.01	0.01	0.01	0.01	0.00	0.01	0.02	3
MEMIT	0.78	0.76	0.53	0.69	0.91	**0.89**	0.50	0.77	**0.98**	**0.92**	0.76	**0.89**	0.78	3
DEFER	0.63	0.58	0.62	0.61	0.37	0.36	**1.00**	0.58	0.34	0.32	0.85	0.50	0.56	3
WISE	0.84	0.78	**0.99**	0.87	0.68	0.64	0.99	0.77	0.76	0.68	**1.00**	0.81	0.82	3
ONCEEDIT	**0.99**	**0.92**	**0.99**	**0.97**	**0.95**	0.88	0.98	**0.93**	0.84	0.76	0.99	0.87	**0.92**	1
Counterfact														
FT-L	0.26	0.01	0.18	0.15	0.41	0.05	**0.99**	0.48	0.71	0.09	0.07	0.30	0.31	3
MEND	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	3
ROME	0.07	0.04	0.05	0.05	0.01	0.02	0.00	0.01	0.00	0.00	0.00	0.00	0.02	3
MEMIT	0.95	0.51	0.23	0.56	0.78	0.43	0.26	0.49	**0.99**	0.20	**0.90**	0.70	0.58	3
DEFER	0.98	**0.88**	0.34	0.74	0.47	0.53	0.79	0.60	0.94	**0.84**	0.15	0.64	0.66	3
WISE	0.74	0.33	0.38	0.48	0.67	0.24	0.35	0.42	0.37	0.08	0.37	0.27	0.39	3
ONCEEDIT	**0.99**	0.81	**0.62**	**0.81**	**0.94**	**0.76**	0.62	**0.77**	0.94	0.76	0.53	**0.74**	**0.72**	1

Table 1: Experimental results on ZsRE and Counterfact under teacher-forced setting. **Bold** and underline numbers indicate the best and second performance among evaluated methods. Score represents the average output of the three models (Rel., Gen., and Loc.), while Freq. indicates the total number of edits required to update these models.

the partial order relations in the aggregated distribution, ultimately leading to a more effective representation of the new knowledge.

4 Experiments

4.1 Experimental Setups

Datasets Building on previous works (Meng et al., 2022a; Yao et al., 2023), we conduct our experiments using two widely-used model editing datasets: ZsRE (Levy et al., 2017) and Counterfact (Meng et al., 2022a). ZsRE is a context-free question-answering dataset, and we adopt the dataset split following Zhang et al. (2024a). Counterfact is a counterfactual dataset in its completed form, which is employed to assess the impact of model editing techniques on entity-relation triples.

Metrics We evaluate all methods from three perspectives based on the EasyEdit (Wang et al., 2023b), as defined in §2.1: Reliability (Rel.), Generality (Gen.), and Locality (Loc.), which are commonly used in prior works (Wang et al., 2024b; Hartvigsen et al., 2024). The final score is the average accuracy across these three sets.

Baselines We select seven trending baselines compared with ONCEEDIT, covering four distinct types of knowledge editing methods: 1) *Constrained fine-tuning*: **FT-L** (Meng et al., 2022a), focuses on fine-tuning a single layer's FFN with new knowledge while incorporating an additional KL divergence loss. 2) *Locate-then-edit*: **ROME** (Meng et al., 2022a) and **MEMIT** (Meng et al., 2022b), employ causal tracing to identify model areas relevant to the desired edit, followed by targeted updates to the corresponding parameters. 3) *Meta-learning*: **MEND** (Mitchell et al., 2021), trains an external hyper-network to model the gradients produced by conventional fine-tuning. 4) *Memory-based*: This category encompasses the **DEFER** (Hartvigsen et al., 2024), **WISE** (Wang et al., 2024b), and **GRACE** (Hartvigsen et al., 2024) methods, all of which use dedicated memory to store and manage edited knowledge.

Implementation Details We conduct experiments on three popular models from prior research: Llama2-7B (Touvron et al., 2023), Mistral-7B-v0.1 (Jiang et al., 2023a), and GPT-J-6B (Wang and Komatsuzaki, 2021). For the datasets, we sample 1,000 records from the evaluation sets of ZsRE and Counterfact under the batch editing setting, where the evaluation is conducted after all knowledge editing operations have been completed. Meanwhile, we select Tiny-Llama (Zhang et al., 2024b) as the plug-in model for ONCEEDIT. Additionally, we utilize EasyEdit for evaluation, incorporating two decoding strategies: teacher-forced and validation generation. For the main experiment, we apply both strategies, with the teacher-forced strategy being commonly employed in prior research and the validation generation strategy better reflecting real-

Method	Llama2-7B				Mistral-7B-v0.1				GPT-J-6B				Score↑
	Rel.↑	Gen.↑	Loc.↑	Avg.↑	Rel.↑	Gen.↑	Loc.↑	Avg.↑	Rel.↑	Gen.↑	Loc.↑	Avg.↑	
ZsRE													
DEEPEN	0.88	0.81	0.02	0.57	0.73	0.67	0.02	0.47	0.20	0.19	0.14	0.18	0.41
+*DW*	0.88	0.77	0.99	0.88	0.73	0.65	0.95	0.78	0.20	0.19	0.99	0.46	0.71
+*DW*+*EE*(Ours)	0.99	0.89	0.99	0.96	0.93	0.83	0.96	0.91	0.84	0.76	0.99	0.86	0.91
Counterfact													
DEEPEN	0.79	0.60	0.11	0.50	0.64	0.51	0.14	0.43	0.10	0.06	0.11	0.09	0.34
+*DW*	0.81	0.60	0.40	0.60	0.64	0.51	0.41	0.52	0.11	0.07	0.29	0.16	0.43
+*DW*+*EE*(Ours)	0.99	0.82	0.36	0.72	0.94	0.76	0.37	0.69	0.93	0.74	0.23	0.63	0.68

Table 2: Ablation study on the Dynamic Weight (DW) mechanism by `[WEIGHT]` and the Ensemble Enhancement (EE) mechanism which includes *Search-space Zero Initialization* and *Target Augmentation*. Green indicates improved performance compared to the previous row, while gray indicates a decline compared to the previous row.

world scenarios. Among the baselines, GRACE primarily reports results based on the validation generation strategy. Further details are provided in Appendix A.

4.2 Main Results

The main experimental results are shown in Table 1. ONCEEDIT achieves the highest overall scores, surpassing the second-best methods by 14% on ZsRE and 6% on Counterfact. Unlike traditional knowledge editing methods that require separate edits for each model, ONCEEDIT updates multiple models with a single edit, demonstrating its efficiency. Moreover, ONCEEDIT achieves top performance in all five settings except for a slightly lower score on GPT-J-6B with ZsRE, demonstrating its strong capability in single-model editing.

Additionally, the results indicate that other methods exhibit significant performance fluctuations across datasets and models. For instance, MEMIT performs well on GPT-J-6B with Counterfact but poorly on ZsRE (0.92 vs. 0.20), and FT-L shows exceptionally high locality (0.99) on Mistral-7B-v0.1 under Counterfact while underperforming on other models. In contrast, ONCEEDIT maintains stable and consistent results across all models and datasets. Notably, due to the nature of Counterfact completions, all methods yield lower locality on it compared to ZsRE. Despite this, ONCEEDIT achieves comparable locality across different models. Overall, the main experiment demonstrates that ONCEEDIT *enables stable and effective knowledge editing across multiple models and datasets.*

Previous studies typically evaluate methods under teacher-forced conditions. However, the validation generation setting, which relies solely on output generation, more accurately reflects a model's ability to understand and apply the knowledge, and better aligns with real-world scenarios. As a result, we choose to supplement our evaluation with this setting. The experimental results, presented in Figure 3, highlight four methods that stand out in terms of performance. Notably, ONCEEDIT exhibits superior editing capabilities, outperforming all other methods. In contrast, MEMIT, which performs relatively well under teacher-forced, shows a significant drop in performance under the validation generation setting. Although GRACE achieves a strong overall score, it demonstrates poor generalization, a limitation that has also been observed in Wang et al. (2024b). More detailed results can be found in Table 7.

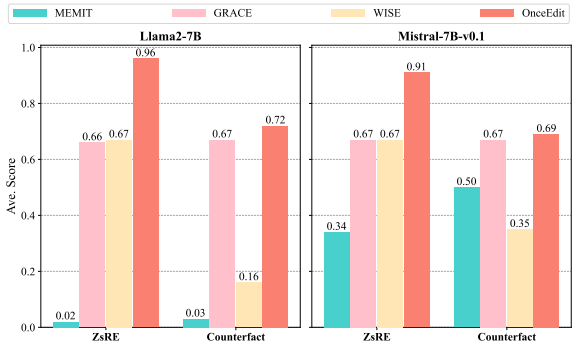

Figure 3: Experimental results on ZsRE and Counterfact under validation generation setting. Score represents the average output of each model (Rel., Gen., and Loc.).

5 Further Analysis and Ablation Study

5.1 Ablation Study

In this section, we conduct a series of ablation studies to evaluate the effectiveness of ONCEEDIT's components and the impact of hyperparameters.

Methods	Llama2-7B	Mistral-7B-v0.1	GPT-J-6B	Score
FT-L	0.23	0.50	0.23	0.32
MEMIT	0.69	0.77	**0.89**	0.78
DEFER	0.61	0.58	0.50	0.56
WISE	0.87	0.77	0.81	0.82
OE(Tiny-Llama)	**0.97**	**0.93**	0.87	**0.92**
OE(Qwen2.5-1.5B)	0.83	0.83	**0.89**	0.85

Table 3: Experimental results on the ZsRE dataset using different plugin models. **OE** represents the ONCEEDIT method. **Bold** and underline numbers indicate the best and second performance among evaluated methods.

Methods	Llama2-7B	Mistral-7B-v0.1	GPT-J-6B	Total
FT-L	0.69	0.71	0.73	2.13
MEND	-	-	-	-
ROME	2.29	3.39	2.50	8.18
MEMIT	-	-	-	-
GRACE	0.65	0.72	0.84	2.21
DEFER	1.49	1.47	1.40	4.36
WISE	1.35	1.33	1.26	3.94
ONCEEDIT	1	1	1	1

Table 4: The editing times for each method are normalized relative to the time taken by ONCEEDIT, with '-' indicating a time that is more than 100 times longer. The term 'total' refers to the overall time required to edit all three models.

Models	TFLOPS/token	TFLOPS_{all}	Num
Llama2-7B	0.013	12.638	973
Mistral-7B-v0.1	0.014	9.552	683
GPT-J-6B	0.011	9.739	886

Table 5: Analysis of the TFLOPS computational cost for model ensemble attributed to the relative representation matrix across three models.

Components Ablation. We examine the impact of the mechanisms introduced in §3 on 1,000 edited instances under validation generation. The results are presented in Table 2. ONCEEDIT progressively enhances the overall editing performance of the three models by incorporating the dynamic weight mechanism and the ensemble enhancement mechanism into DEEPEN. When applied to a single model on the datasets, the method with both mechanisms also achieves the best performance.

To evaluate the impact of these two mechanisms separately, we analyze their effects on model performance across different settings. For the dynamic weight mechanism, results indicate that it significantly enhances locality while maintaining reliability and generality in most cases. The only notable drawback is a minor generalization loss of 2% to 4% on Llama2-7B and Mistral-7B-v0.1 on ZsRE, which remains within an acceptable range. Meanwhile, the ensemble enhancement mechanism proves to be highly beneficial, substantially improving reliability and generality across all settings in line with our design objectives. While it introduces a 4% to 6% reduction in locality for the three models on the Counterfact dataset, this trade-off is outweighed by gains in the other two aspects. Notably, the overall average score of each model on Counterfact increases by more than 12%, making the slight decrease in locality an acceptable compromise.

Hyperparameters Ablation. We conduct an ablation study on the balanced hyperparameter λ for multi-task learning during the editing stage. Specifically, we select 200 edited instances and evaluate the performance under five different λ settings. The experimental results, presented in Table 8, indicate that λ is relatively robust. With the exception of the extreme case where $\lambda = 0$, all other settings yield good editing performance.

Plugin Models. In the main experiments, we selected Tiny-Llama as the plugin model, as it is one of the most widely used lightweight models. Moreover, in practical applications, OnceEdit demonstrates high adaptability, enabling seamless integration with various LLMs through a unified plugin model. This design supports flexible knowledge updates without requiring a switch between different plugin models.

However, to further explore the performance of alternative plugin models, we conducted additional experiments on the ZsRE dataset using Qwen2.5-1.5B as the plugin model, integrated with three different LLMs. We compared its overall knowledge editing performance against strong baselines, including FT-L, MEMIT, WISE, and DEFER, as presented in Table 3.

The results show that the ONCEEDIT method, when using Qwen2.5-1.5B as a plugin model, achieves a higher overall score than all baselines, further validating the effectiveness of OnceEdit. Its performance is only second to TinyLlama, which may be influenced by the post-editing performance of the plugin model itself. In practical applications, selecting a more stable small model (e.g., TinyLlama) as the plugin model can lead to more effective knowledge editing.

5.2 Cost of Editing and Ensemble

In this section, we quantitatively evaluate the editing time required for each method under identical

hardware conditions and dataset scales.

The editing time statistics are summarized in Table 4. MEMIT involves computing second-order momentum to ensure locality, while MEND requires training an additional hypernetwork using a training set. Due to the substantial computational cost of these two methods, their results are omitted from the table. FT-L incurs lower time overhead for single-model editing as it only updates a specific layer of the model. Similarly, GRACE maintains a working memory for a specific layer, resulting in relatively low editing time as well. However, as the number of models to be edited increases, the efficiency of ONCEEDIT becomes increasingly evident. When editing three models, the editing time for other methods is more than twice that of ONCEEDIT. This demonstrates that ONCEEDIT is highly efficient in multi-model knowledge editing scenarios, with its advantages becoming more pronounced as the number of models increases.

In addition, we analyze the migration cost of ONCEEDIT between LLMs, which mainly comes from the calculation of the relative transfer matrix of the integrated models. We combined DEEPEN to derive the FLOPS formula required to calculate the relative transfer matrices for the plug-in model and LLMs. The detailed derivation process is shown in the Appendix C. We calculate the total FLOPS, denoted as TFLOPS_{all}, for the relative transfer matrix required for each pair of integrated models based on Equations 25 and 26. The results are summarized in Table 5. Specifically, for the three models being edited, the cost of constructing the relative transfer matrices is equivalent to generating approximately 600 to 1000 tokens during forward propagation. This overhead is modest compared to the cost of performing another round of editing, demonstrating that the migration overhead introduced by our model integration is acceptable.

5.3 Extending on More Models

In the main experiment, we aim to provide an effective comparison with other popular knowledge editing methods by following previous studies and selecting three classic models. Unlike other methods, which are often sensitive to hyperparameters, ONCEEDIT demonstrates strong adaptability and can be quickly generalized to new models. To further validate this, we selected four newer, larger, and more diverse models, including Llama3-8B, Mistral-7B-v0.3, Qwen2.5-7B, and Llama3-70B, and applied ONCEEDIT to edit 200 instances under the validation generation setting. The results, as shown in Table 6, highlight ONCEEDIT's ability to achieve effective and stable editing across all four models and two datasets, further confirming its high scalability.

Model	Rel.↑	Gen.↑	Loc.↑	Avg.↑
ZsRE				
Llama3-8B	0.91	0.90	0.97	0.93
Mistral-7B-v0.3	0.93	0.93	0.95	0.94
Qwen2.5-7B	0.90	0.90	0.74	0.85
Llama3-70B	0.74	0.75	0.91	0.80
Counterfact				
Llama3-8B	0.99	0.91	0.16	0.69
Mistral-7B-v0.3	0.95	0.87	0.25	0.69
Qwen2.5-7B	0.98	0.92	0.18	0.70
Llama3-70B	0.79	0.72	0.18	0.56

Table 6: Extended experimental results on editing 200 instances across multiple models using ONCEEDIT.

6 Related Work

Knowledge Editing. Knowledge editing (Yao et al., 2023; Feng et al., 2023) is an effective compensatory approach for updating models' knowledge, categorized into four main types: fine-tuning, locate-then-edit, meta-learning, and memory-based methods. Constrained fine-tuning (Meng et al., 2022a), while straightforward for correcting model behavior, often risks damaging non-edited knowledge. Locate-then-edit methods typically involve identifying and updating specific parameters, as seen in ROME (Meng et al., 2022a), which uses MLP-based memories for factual edits, and MEMIT (Meng et al., 2022b), which extends this to batch edits. Meta-learning approaches, such as MEND (Mitchell et al., 2021), involve training an external hyper-network to predict updates to the original model. MALMEN (Tan et al., 2023) further addresses the issue of the cancellation effect in MEND by framing parameter shift aggregation as a least-squares problem. Memory-based methods like SERAC (Mitchell et al., 2022) and GRACE (Hartvigsen et al., 2024) employ working memories to store edits, dynamically selecting parameters based on input similarity. Furthermore, recent studies (Wang et al., 2024b; Wang and Li, 2024a) have further investigated ways to reduce the adverse effects associated with sequential edits. While existing approaches predominantly focus on individual

models, there is a lack of research addressing the efficient editing of multiple models.

Model Ensemble. The model ensemble approach (Lu et al., 2024) integrates the strengths of multiple models to produce refined answers and can be categorized by fusion granularity into output-level and probability-level ensembles (Yao et al., 2024). In output-level model ensembles, the outputs from multiple models are combined as candidate sets. Methods like PAIRRANKER (Jiang et al., 2023b) and routing mechanisms (Lu et al., 2023) select the best candidate based on pairwise comparison or input-specific suitability. Other studies (Wang et al., 2023a; Jiang et al., 2023b) train fusion modules to integrate outputs effectively. Probability-level model ensembles, on the other hand, focus on merging the probability distributions of multiple models at each decoding step. This process is particularly challenging when dealing with heterogeneous models due to the need for vocabulary alignment. To address this issue, EVA (Xu et al., 2024) employs overlapping tokens to learn token alignment across different vocabularies, while DEEPEN (Huang et al., 2024b) transforms the representations of each model into a shared space using common vocabulary tokens.

7 Conclusion

In this work, we introduce OnceEdit which addresses the challenges of efficiency and stability in multi-model editing scenarios. By leveraging a lightweight plug-in model as the editing module and employing improved heterogeneous model ensemble techniques, OnceEdit enables knowledge updates across multiple models with low migration costs. Extensive evaluations across multiple models and datasets demonstrate that OnceEdit outperforms existing knowledge editing methods in both teacher-forced and validation generation settings. Further analysis confirms OnceEdit's adaptability and stability, underscoring its potential as an effective solution for real-world scenarios.

Limitations

Despite OnceEdit's high efficiency and adaptability, there are several limitations worth noting. Firstly, although the plug-in model is relatively small and inference speed can be improved through strategies such as parallel decoding, OnceEdit inevitably incurs additional overhead due to the inclusion of the plug-in model. Secondly, our experiments focused exclusively on the batch editing setting and did not explore more complex scenarios, such as sequential editing or multi-hop editing tasks. Thirdly, our study primarily adopted a direct fine-tuning approach for knowledge editing within the plug-in model. It is important to emphasize that our framework is fundamentally orthogonal to existing knowledge editing methods. In future work, we plan to enhance the plug-in model by integrating more advanced editing techniques, enabling us to more effectively address challenges such as sequential editing and generalization.

Acknowledgements

Xiaocheng Feng is the corresponding author of this work. We thank the anonymous reviewers for their insightful comments. This work was supported by the National Natural Science Foundation of China (NSFC) (grant 62276078, U22B2059), the Key R&D Program of Heilongjiang via grant 2022ZX01A32, and the Fundamental Research Funds for the Central Universities (Grant No.HIT.OCEF.2023018).

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Zhangyin Feng, Weitao Ma, Weijiang Yu, Lei Huang, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. 2023. Trends in integration of knowledge and large language models: A survey and taxonomy of methods, benchmarks, and applications. *arXiv preprint arXiv:2311.05876*.

Tom Hartvigsen, Swami Sankaranarayanan, Hamid Palangi, Yoon Kim, and Marzyeh Ghassemi. 2024. Aging with grace: Lifelong model editing with discrete key-value adaptors. *Advances in Neural Information Processing Systems*, 36.

Lei Huang, Xiaocheng Feng, Weitao Ma, Yuchun Fan, Xiachong Feng, Yangfan Ye, Weihong Zhong, Yuxuan Gu, Baoxin Wang, Dayong Wu, et al. 2025. Improving contextual faithfulness of large language models via retrieval heads-induced optimization. *arXiv preprint arXiv:2501.13573*.

Lei Huang, Xiaocheng Feng, Weitao Ma, Liang Zhao, Yuchun Fan, Weihong Zhong, Dongliang Xu, Qing Yang, Hongtao Liu, and Bing Qin. 2024a. Advancing large language model attribution through self-improving. *arXiv preprint arXiv:2410.13298*.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. 2023. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *arXiv preprint arXiv:2311.05232*.

Yichong Huang, Xiaocheng Feng, Baohang Li, Yang Xiang, Hui Wang, Bing Qin, and Ting Liu. 2024b. Enabling ensemble learning for heterogeneous large language models with deep parallel collaboration. *arXiv preprint arXiv:2404.12715*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023a. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. 2023b. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion. *arXiv preprint arXiv:2306.02561*.

Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. 2017. Zero-shot relation extraction via reading comprehension. *arXiv preprint arXiv:1706.04115*.

Junyi Li, Jie Chen, Ruiyang Ren, Xiaoxue Cheng, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. 2024a. The dawn after the dark: An empirical study on factuality hallucination in large language models. *arXiv preprint arXiv:2401.03205*.

Shuaiyi Li, Yang Deng, Deng Cai, Hongyuan Lu, Liang Chen, and Wai Lam. 2024b. Consecutive model editing with batch alongside hook layers. *arXiv preprint arXiv:2403.05330*.

Jinliang Lu, Ziliang Pang, Min Xiao, Yaochen Zhu, Rui Xia, and Jiajun Zhang. 2024. Merge, ensemble, and cooperate! a survey on collaborative strategies in the era of large language models. *arXiv preprint arXiv:2407.06089*.

Keming Lu, Hongyi Yuan, Runji Lin, Junyang Lin, Zheng Yuan, Chang Zhou, and Jingren Zhou. 2023. Routing to the expert: Efficient reward-guided ensemble of large language models. *arXiv preprint arXiv:2311.08692*.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022a. Locating and editing factual associations in gpt. *Advances in Neural Information Processing Systems*, 35:17359–17372.

Kevin Meng, Arnab Sen Sharma, Alex Andonian, Yonatan Belinkov, and David Bau. 2022b. Mass-editing memory in a transformer. *arXiv preprint arXiv:2210.07229*.

AI Meta. 2024. Introducing meta llama 3: The most capable openly available llm to date. *Meta AI Blog (accessed 2024–04–20). There is no corresponding record for this reference*.

Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D Manning. 2021. Fast model editing at scale. *arXiv preprint arXiv:2110.11309*.

Eric Mitchell, Charles Lin, Antoine Bosselut, Christopher D Manning, and Chelsea Finn. 2022. Memory-based model editing at scale. In *International Conference on Machine Learning*, pages 15817–15831. PMLR.

Chenmien Tan, Ge Zhang, and Jie Fu. 2023. Massive editing for large language models via meta learning. *arXiv preprint arXiv:2311.04661*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Ben Wang and Aran Komatsuzaki. 2021. Gpt-j-6b: A 6 billion parameter autoregressive language model.

Hongyi Wang, Felipe Maia Polo, Yuekai Sun, Souvik Kundu, Eric Xing, and Mikhail Yurochkin. 2023a. Fusing models with complementary expertise. *arXiv preprint arXiv:2310.01542*.

Mengru Wang, Yunzhi Yao, Ziwen Xu, Shuofei Qiao, Shumin Deng, Peng Wang, Xiang Chen, Jia-Chen Gu, Yong Jiang, Pengjun Xie, et al. 2024a. Knowledge mechanisms in large language models: A survey and perspective. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 7097–7135.

Peng Wang, Zexi Li, Ningyu Zhang, Ziwen Xu, Yunzhi Yao, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. 2024b. Wise: Rethinking the knowledge memory for lifelong model editing of large language models. *arXiv preprint arXiv:2405.14768*.

Peng Wang, Ningyu Zhang, Xin Xie, Yunzhi Yao, Bozhong Tian, Mengru Wang, Zekun Xi, Siyuan Cheng, Kangwei Liu, Guozhou Zheng, et al. 2023b. Easyedit: An easy-to-use knowledge editing framework for large language models. *arXiv preprint arXiv:2308.07269*.

Renzhi Wang and Piji Li. 2024a. Lemoe: Advanced mixture of experts adaptor for lifelong model editing of large language models. *arXiv preprint arXiv:2406.20030*.

Renzhi Wang and Piji Li. 2024b. Memoe: Enhancing model editing with mixture of experts adaptors. *arXiv preprint arXiv:2405.19086*.

Yangyifan Xu, Jinliang Lu, and Jiajun Zhang. 2024. Bridging the gap between different vocabularies for llm ensemble. *arXiv preprint arXiv:2404.09492*.

Yunzhi Yao, Peng Wang, Bozhong Tian, Siyuan Cheng, Zhoubo Li, Shumin Deng, Huajun Chen, and Ningyu Zhang. 2023. Editing large language models: Problems, methods, and opportunities. *arXiv preprint arXiv:2305.13172*.

Yuxuan Yao, Han Wu, Mingyang Liu, Sichun Luo, Xiongwei Han, Jie Liu, Zhijiang Guo, and Linqi Song. 2024. Determine-then-ensemble: Necessity of top-k union for large language model ensembling. *arXiv preprint arXiv:2410.03777*.

Ningyu Zhang, Yunzhi Yao, Bozhong Tian, Peng Wang, Shumin Deng, Mengru Wang, Zekun Xi, Shengyu Mao, Jintian Zhang, Yuansheng Ni, et al. 2024a. A comprehensive study of knowledge editing for large language models. *arXiv preprint arXiv:2401.01286*.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024b. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*.

Zihan Zhang, Meng Fang, Ling Chen, Mohammad-Reza Namazi-Rad, and Jun Wang. 2023. How do large language models capture the ever-changing world knowledge? a review of recent advances. *Preprint*, arXiv:2310.07343.

Weihong Zhong, Xiaocheng Feng, Liang Zhao, Qiming Li, Lei Huang, Yuxuan Gu, Weitao Ma, Yuan Xu, and Bing Qin. 2024. Investigating and mitigating the multimodal hallucination snowballing in large vision-language models. *arXiv preprint arXiv:2407.00569*.

A Implementation Details

Our experiment evaluates ONCEEDIT under batch editing by comparing it with seven knowledge editing methods: FT-L, MEND, ROME, MEMIT, DEFER, GRACE, and WISE. All experiments were conducted on NVIDIA A100 80GB GPUs. The hyperparameter settings for these baselines follow previous works (Wang et al., 2024b; Wang and Li, 2024a). Below, we provide an overview of these methods along with their implementation details:

- **FT-L** is a constrained fine-tuning approach that updates only a single MLP layer while imposing a L_∞ norm constraint on weight modifications. In our experiments, we select the 21-th layer for GPT-J-6B and the 27-th layer for Llama2-7B and Mistral-7B-v0.1. The fine-tuning learning rate is set to 5e-4. Notably, to avoid OOM issues, we adopt a batch-based strategy, where we use 5 batches, each updating 200 knowledge entries per round when testing FT-L.
- **MEND** is a meta-learning-based approach that trains an external hyper-network to simulate gradients. It employs low-rank decomposition with a specialized design to reduce the size of the hyper-network. For the training phase of MEND, we align the experimental settings entirely with those used in EasyEdit (Wang et al., 2023b).
- **ROME** locates layers relevant to edits by first disrupting and then restoring activations. It subsequently updates the parameters of feedforward networks (FFNs) in a direct manner to modify knowledge. We select the [3,4,5,6,7,8] layers as the target layer for the GPT-J-6B, and [4,5,6,7,8] for the Llama2-7B and Mistral-7B-v0.1.
- **MEMIT** utilizes the same knowledge localization method as ROME but enables simultaneous updates across multiple layers, allowing for the batch integration of thousands of edited knowledge entries. Consequently, MEMIT and ROME share the same target layer selection.
- **DEFER** is a reimplementation of SERAC, utilizes an external memory to store editing instances and trains an additional scope classifier and counterfactual model to appropriately respond to inputs. we set the learning rate is 7e-5 and select the 21-th layer for GPT-J-6B and the 27-th layer for Llama2-7B and Mistral-7B-v0.1.
- **GRACE** leverages a discrete key-value codebook to perform knowledge editing. Throughout the editing process, the codebook is dynamically maintained by introducing new keys, expanding existing ones, and splitting them as needed. During inference, the method identifies the closest matching key and determines whether to adjust the activation of the hidden layer output. For the learning rate and ϵ_{init}, we also align the experimental settings with those used in EasyEdit.
- **WISE** stores different edits in separate side memories and routes input queries to the appropriate memory based on activation scores. For the hyperparameters setting of WISE, we adhere to the original paper.

Apart from the aforementioned methods, there are also two more recent approaches, MEMoe (Wang and Li, 2024b) and LEMoE (Wang and Li, 2024a), whose results are not included due to the unavailability of their source code.

For ONCEEDIT, we use Tiny-Llama as the plug-in model, setting the learning rate to 1.0e-4 and the multi-task balanced hyperparameter $\lambda = 0.8$. Since both ONCEEDIT and WISE require the introduction of unrelated knowledge as auxiliary information, we select the instances from the training sets of ZsRE and Counterfact. Additionally, due to the edited models are the base model, we incorporate prompt-assisted fine-tuning for the ZsRE and Counterfact. The prompts are as follows: For ZsRE: "Answer this question:\n[Question]: {Input}\n[Answer]:". For Counterfact: "Complete this half sentence.\n [Half Sentence]: {Input}\n[Answer]:" To ensure a fair comparison, we evaluate the baselines both with and without prompts, reporting the highest observed value.

B Evalution for Knowledge Editing

Following prior research, an effective knowledge editing method should satisfy three essential properties: **Reliability**, **Generality**, and **Locality**. These properties serve as important evaluation metrics for editing methods.

Reliability refers to the model's ability to correctly respond to inputs from the edited set. Specifically, the edited model θ' should consistently produce the correct output for the instances in S_E:

$$\mathbb{E}_{(x_e,y_e)\in S_E} \mathbb{1}\left\{f_{\theta'}\left(x_e\right) = y_e\right\}. \tag{12}$$

Generality refers to the edited model's capacity to apply the edited knowledge beyond the specific examples in S_E. Specifically, the model should be able to correctly respond to the instances in the set S_R, where x_r is a rephrased version of an edit x_e,

and the expected output remains y_e:

$$\mathbb{E}_{(x_r,y_e)\in S_R} \mathbb{1}\left\{f_{\theta'}\left(x_r\right) = y_e\right\}. \quad (13)$$

Locality emphasizes that the edited model should not alter its behavior on the non-edited knowledge. Specifically, for instances in the dataset S_L, which are not affected by the edits, the edited model should produce the same output as before the edit:

$$\mathbb{E}_{(x_{loc},y_{loc})\in S_L} \mathbb{1}\left\{f_{\theta'}\left(x_{loc}\right) = f_{\theta}\left(x_{loc}\right)\right\}. \quad (14)$$

Method	Llama2-7B				Mistral-7B-v0.1			
	Rel.↑	Gen.↑	Loc.↑	Avg.↑	Rel.↑	Gen.↑	Loc.↑	Avg.↑
ZsRE								
MEMIT	0.03	0.03	0.00	0.02	0.50	0.48	0.04	0.34
WISE	0.58	0.46	**0.99**	0.67	0.57	0.46	0.99	0.67
GRACE	**1.00**	0.00	**0.99**	0.66	**0.98**	0.03	**1.00**	0.67
ONCEEDIT	0.99	**0.89**	**0.99**	**0.96**	0.93	**0.83**	0.96	**0.91**
Counterfact								
MEMIT	0.03	0.03	0.03	0.03	0.79	0.72	0.01	0.50
WISE	0.34	0.04	0.09	0.16	0.33	0.08	0.65	0.35
GRACE	**1.00**	0.00	**0.99**	0.67	**0.99**	0.00	**0.99**	0.67
ONCEEDIT	0.99	**0.82**	0.36	**0.72**	0.94	**0.76**	0.37	**0.69**

Table 7: Experimental results on ZsRE and Counterfact under validation generation setting. **Bold** and underline numbers indicate the best and second performance among evaluated methods.

Lambda	Llama2-7B				Mistral-7B-v0.1			
	Rel.↑	Gen.↑	Loc.↑	Avg.↑	Rel.↑	Gen.↑	Loc.↑	Avg.↑
ZsRE								
0.00	0.05	0.04	0.99	0.36	0.05	0.05	0.98	0.36
0.20	0.99	0.95	0.99	0.98	0.94	0.90	0.98	0.94
0.40	0.99	0.97	1.00	**0.99**	0.94	0.92	0.99	**0.95**
0.60	0.99	0.95	0.99	0.98	0.94	0.91	0.98	0.94
0.80	0.99	0.98	1.00	**0.99**	0.93	0.93	0.98	**0.95**
1.00	0.99	0.97	1.00	**0.99**	0.94	0.91	0.99	**0.95**
Counterfact								
0.00	0.00	0.00	0.99	0.33	0.01	0.00	0.96	0.32
0.20	0.99	0.84	0.68	**0.83**	0.94	0.80	0.67	**0.80**
0.40	0.99	0.70	0.76	0.82	0.95	0.72	0.66	0.78
0.60	0.99	0.80	0.71	**0.83**	0.95	0.77	0.70	**0.80**
0.80	0.99	0.89	0.53	0.80	0.95	0.85	0.49	0.76
1.00	0.95	0.84	0.67	**0.83**	0.95	0.80	0.66	**0.80**

Table 8: Ablation study on the impact of lambda (λ) during the editing stage, evaluated on 200 edited instances under validation generation. **Bold** numbers indicate the best performance among diverse settings.

C FLOPS of Relative Transfer Matrix

In this section, we derive the statistical formula for the FLOPS required by ONCEEDIT to compute the relative transfer matrix during the ensemble stage. Specifically, given an LLM parametrized by θ, and a tiny model parametrized by $\tilde{\theta}$, let the vocabulary of LLM be M and that of the small model be N. The dimension of embeddings in LLM is denoted as d_l, while that of small model is d_s. Additionally, we define A as the anchor tokens set shared between the two models.

Following DEEPEN (Huang et al., 2024b), we compute the relative transfer matrices for both the LLM and the tiny model. Here, we take the LLM matrix R_l as an example, while the derivation for the tiny model follows analogously. Formally, the relative representation matrix $R_l \in \mathbb{R}^{|M|\times|A|}$ encodes the relative representation of each word $m^{(i)}$ in the LLM's vocabulary. The i-th row of R_l is given by:

$$R_l[i] = (cos(e_{m^{(i)}}, e_{a^{(1)}}), ..., cos(e_{m^{(i)}}, e_{a^{(|\mathbb{A}|)}})), \quad (15)$$

where $e_{m^{(i)}}$ and $e_{a^{(1)}}$ denote the embeddings of word $m^{(i)}$ and $a^{(j)}$, respectively.

To address the representation degeneration of outlier words, DEEPEN applies a softmax normalization to transform the relative representations into a probability distribution:

$$\hat{R}_l[i] = \text{softmax}(R_l[i]). \quad (16)$$

For Equation 15, the cosine similarity between each vocabulary word and the anchor tokens is computed as:

$$R_l[i,j] = \frac{E_l[i] \cdot A_l[j]^T}{\|E_l[i]\|\|A_l[j]\|}, \quad (17)$$

where $E_l \in \mathbb{E}^{|M|\times|d_l|}$ and $A_l \in \mathbb{E}^{|A|\times|d_l|}$ represents the original word embedding matrix and the anchor words word embedding matrix of the LLM, respectively.

The L2 norm of E_l and A_l is calculated as follows:

$$\|E_l[i]\| = \sqrt{\sum_{j=1}^{d_l} E_[i,j]^2}, \quad (18)$$

$$\|A_l[i]\| = \sqrt{\sum_{j=1}^{d_l} A_[i,j]^2}. \quad (19)$$

Based on Equation 17, 18 and 19, the total FLOPS required to calculate R_l consists of three components:the dot product operation, the L2 norm computation, and the final division. These are formulated as follows:

$$\text{FLOPS}_{Dot} = |M| * |A| * (d_l + d_l - 1), \quad (20)$$

Models	Vocab. Size	AW Num	Embedding Dim.
Llama2-7B	32000	31999	4096
Mistral-7B-v0.1	32000	24184	4096
GPT-J-6B	50400	17830	4096
Tiny-Llama	32001	-	2048

Table 9: configurations for calculating relative transfer matrix FLOPS. Vocab. Size denotes the vocabulary size, AW Num represents the number of anchor words shared between the models and Tiny-Llama, and Embedding Dim. refers to the dimension of the embedding layer.

$$\begin{aligned}\text{FLOPS}_{L2} &= |M| * (d_l + d_l - 1 + C_{sqrt}) \\ &+ |A| * (d_l + d_l - 1 + C_{sqrt}) + 1,\end{aligned} \tag{21}$$

$$\text{FLOPS}_{Div} = |M| * |A|, \tag{22}$$

where FLOPS_{Dot}, FLOPS_{L2} and FLOPS_{Div} correspond to the FLOPS of the dot product, L2 norm, and division operations, respectively. The term C_{sqrt} represents the computational cost of the square root operation, which is a constant.

For the computed R_l, a softmax operation is applied for normalization, defined as follows:

$$\text{softmax}(R_l[i,j]) = \frac{\exp(R_l[i,j])}{\sum_{j=1}^{|A|} \exp(R_l[i,j])}. \tag{23}$$

The FLOPS required for the softmax computation on R_l are given by:

$$\begin{aligned}\text{FLOPS}_{softmax} &= |M| * |A| * C_{exp} \\ &+ |M| * (|A| - 1) + |M| * |A|,\end{aligned} \tag{24}$$

where C_{exp} represents the computational cost of the exponential operation, which is a constant.

In summary, the total FLOPS, denoted as FLOPS^l_{all}, required to compute the relative transfer matrix of LLM is the sum of the computational costs from the Equation 17, 18 ,19 and 24. For simplification, we set $C_{sqrt} = 2\,\text{FLOPS}$ and $C_{exp} = 25\,\text{FLOPS}$. Under this assumption, the FLOPS^l_{all} is given by:

$$\begin{aligned}\text{FLOPS}^l_{all} &= \text{FLOPS}_{Dot} + \text{FLOPS}_{L2} \\ &+ \text{FLOPS}_{Div} + \text{FLOPS}_{softmax} \\ &= (2 * d_l + 27) * |M| * |A| + 1 \\ &+ 2 * d_l * |M| + (2 * d_l + 1) * |A|.\end{aligned} \tag{25}$$

Based on the reasoning process above, We derive the FLOPS equation for the relative transfer matrix of the tiny model:

$$\begin{aligned}\text{FLOPS}^t_{all} &= (2 * d_s + 27) * |N| * |A| + 1 \\ &+ 2 * d_s * |N| + (2 * d_s + 1) * |A|.\end{aligned} \tag{26}$$

Finally, by integrating Equations 25 and 26 along with the configurations of each model, as summarized in Table 9, we compute the final FLOPS_{all}.